OpenZeppelin | security

OpenZeppelin Contracts Release v5.2 Audit

December 4, 2024

Table of Contents

Summary

Type	Library
Timeline	From 2024-10-21 To 2024-11-06
Languages	Solidity

Total Issues	20 (19 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	10 (9 resolved)
Notes & Additional Information	7 (7 resolved)
Client Reported Issues	1 (1 resolved)

Scope

We audited the OpenZeppelin/openzeppelin-contracts repository at commit 98d28f9. The following files were in scope:

```
contracts/
├── account
│   └── utils
│       ├── draft-ERC4337Utils.sol
│       └── draft-ERC7579Utils.sol
├── governance
│   ├── Governor.sol
│   ├── extensions
│   │   ├── GovernorCountingOverridable.sol
│   │   └── GovernorPreventLateQuorum.sol
│   └── utils
│       └── VotesExtended.sol
├── interfaces
│   ├── draft-IERC4337.sol
│   └── draft-IERC7579.sol
├── proxy
│   └── Clones.sol
├── token
│   └── ERC20
│       ├── extensions
│       │   └── ERC1363.sol
│       └── utils
│           └── ERC1363Utils.sol
└── utils
    ├── Bytes.sol
    ├── CAIP10.sol
    ├── CAIP2.sol
    ├── NoncesKeyed.sol
    └── Strings.sol
```

System Overview

Version 5.2 of the OpenZeppelin Contracts library introduces new utilities designed to facilitate account abstraction and cross-chain messaging, along with a new counting module in the governance contracts.

Account Abstraction

This release includes utility functions for ERC-4337 — the most widely adopted standard for account abstraction — as well as ERC-7579, a newer standard for minimal, modular smart accounts. The `ERC4337Utils` library offers functionality for handling the validation data returned by accounts during signature validation, for hashing user operations, and for unpacking specific values from the `PackedUserOperation` struct defined in ERC-4337.

In contrast, the `ERC7579Utils` library provides functionality for encoding, decoding, and executing various call types defined in the standard, including single, batch, and delegate calls. It also includes constants and custom types that simplify working with the packed execution mode specified in ERC-7579. Additionally, this release includes all interfaces necessary to integrate with these standards.

Cross-Chain Messaging

Chain Agnostic Improvement Proposals (CAIPs) describe standards for blockchain projects that are not specific to a single chain. In particular, CAIP-2 defines a way to uniquely identify a blockchain (e.g., Ethereum, Bitcoin, Cosmos Hub) in a human-readable, developer-friendly and transaction-friendly way, while CAIP-10 defines a way to identify an account in any blockchain specified by a CAIP-2 blockchain ID. This is useful for both decentralized applications and wallets to communicate user accounts or smart contracts across multiple chains using string identifiers specific to each chain. To parse and format identifiers as defined by these CAIPs, the `CAIP2` and `CAIP10` libraries were introduced in this release.

Governance

The governance framework has been enhanced with the introduction of the `_tallyUpdated internal virtual` hook. This hook is designed to be called whenever a proposal's vote tally is updated, providing a customizable point for executing additional logic in response to changes in vote counts. The `GovernorPreventLateQuorum` contract was slightly refactored to leverage the `_tallyUpdated` hook to address scenarios where the quorum is reached late in the voting period. By detecting when the tally update results in a quorum being achieved, it can extend the voting deadline, ensuring that all stakeholders have ample opportunity to participate. This logic was previously placed in a `_castVote` override.

Motivated by the need to accommodate more complex voting scenarios, the `GovernorCountingOverridable` module introduces functionality that allows delegators to override the votes of their delegates. This module, which necessitated the introduction of the `_tallyUpdated` hook, supports two `internal` count functions to manage both standard and override votes. To support the override counting module, the `VotesExtended` contract extends the `Votes` contract by adding checkpoints for delegations and voting units over time. This feature is essential for the `GovernorCountingOverridable` module, as it relies on historical data to accurately process vote overrides and ensure the governance system's integrity.

Clones

The `Clones` library has been enhanced to support the deployment of minimal proxies with immutable arguments. These arguments are provided through a bytes array and appended to the bytecode before deployment through `CREATE` or `CREATE2`. The arguments can be fetched in the implementation contract using the `fetchCloneArgs` function, which is a cheaper alternative to using storage for instance-specific data that can or must be immutable.

Utils

In the `Strings` library, new functions have been added to parse both signed and unsigned integers, hex strings, as well as Ethereum addresses. In addition, a `Bytes` library was created to support byte-specific operations, such as finding the index of a specific byte within a buffer or creating slices from a buffer. These byte operations are designed to mimic the behavior of their JavaScript counterparts, providing developers with a familiar interface.

The `NoncesKeyed` contract is an abstract extension of the `Nonces` contract, designed to support keyed nonces in accordance with the ERC-4337 semi-abstracted nonce system. It implements a mapping structure to manage nonces on a per-key basis for each address, offering functions to retrieve and increment nonces. This contract allows for differentiated nonce management, accommodating scenarios where transactions might require distinct nonce spaces identified by keys.

Lastly, the `ERC1363` contract has been refactored by moving its post-transfer and post-approval callback logic into a new `ERC1363Utils` library.

Security Model and Trust Assumptions

Auditing libraries requires a shift in focus due to their composability within blockchain protocols. While the scope of an audit is typically limited to the code itself, the scope expands when it comes to libraries because of their potential internal and external integrations. Libraries act as foundational components for many protocols. This means that their security is influenced not just by their internal robustness, but also by how they are utilized by integrators. As a result, ensuring a library's security involves reviewing the code as well as anticipating its various use cases and integration scenarios.

In addition to the above, the complexity grows because, while a library must accommodate a wide range of potential use cases, the responsibility for secure implementation often falls on developers who integrate it into their projects. These developers must carefully review the security considerations when extending contracts from the library. A library's security risks can multiply depending on how well developers understand and utilize its contracts. Therefore, extra care is necessary to identify and address all potential threats, both direct and indirect, or to document them so that developers are fully aware of the associated security risks.

Medium Severity

M-01 Improper Input Validation Leads to Incorrect Parsing Results

The `Strings` library includes functions to parse integers and addresses from their string representation. However, some of these functions may produce valid results even when given invalid inputs. Specifically:

- The `tryParseAddress` function validates the input length using only the `begin` and `end` parameters, without actually checking the length of the `input` parameter itself. Combined with the fact that `tryParseHexUint` succeeds even when the `end` parameter is greater than the input length, this might cause addresses shorter than the expected length to parse successfully, which contradicts the function's intended specification.
- The `tryParseUint` and `tryParseHexUint` functions return a success status with a `0` value if the `begin` parameter is greater than or equal to the `end` parameter. For example, empty strings (or `0x` hex strings) are successfully parsed as `0`, which contrasts with the expected behavior in other languages like JavaScript, where `parseInt("")` returns `NaN`.

To prevent the parsing of invalid inputs, consider enforcing stricter validation checks on input lengths and parsing bounds.

Update: *Resolved in pull request #5304 at commit d5e388e and pull request #5324 at commit 3fcb9da. The Contracts team stated:*

> *The team decided to fix the `tryParseAddress` together with M-02 by abstracting the implementation of the `tryParseUint`, `tryParseInt`, and `tryParseHexUint` into a private function that does not check for bounds. This way, bounds are checked only when necessary in other functions.*
>
> *Regarding the inconsistency with the `tryParseUint` and `tryParseHexUint` functions. We think this behavior is consistent with the way the EVM defaults to 0 when there is no data, and there is no other representation of a `NaN` in this context, so we are keeping this behavior.*

M-02 Unbounded Memory Access Within Memory-Safe Assembly

The `_unsafeReadBytesOffset` function in the `Strings` library reads a `bytes32` value from a `bytes` array. Since it delegates the bound checking to the caller, it does so within a memory-safe assembly block and without checking bounds. However, none of the other functions in the library that call `_unsafeReadBytesOffset` perform any validation of the offset that is used to read from the bytes array. Specifically, none of the functions validate that the offset falls inside the length of the array, which can lead to incorrect and undefined behavior that cannot easily be discovered by testing. Note that some functions use `_unsafeReadBytesOffset` to read more than one byte from the array, so that has to be accounted for as well when validating the offset.

Consider ensuring that `_unsafeReadBytesOffset` is only called with offsets that fall within the allocated memory of the `buffer`.

Update: *Resolved in pull request #5304 at commit d5e388e and pull request #5324 at commit 3fcb9da. The Contracts team stated:*

> *The functions using `_unsafeReadBytesOffset` (`tryParseUint`, `tryParseInt` and `tryParseHexUint`) were split into a private version that does not check for bounds, so that it is used when these can be assumed safe.*

Low Severity

L-01 Inconsistent Use of `MCOPY` Opcode

The `MCOPY` opcode was introduced with the Cancun chain upgrade, enabling the copying of one memory space to another. However, since the opcode is still relatively new and may not be supported across all chains, the `_cloneCodeWithImmutableArgs` function of the `Clones` library has been implemented using `abi.encodePacked`. Yet, despite this effort for compatibility, the `slice` function of the `Bytes` library still utilizes this opcode.

Consider clarifying whether the 5.2 release should use the `MCOPY` opcode.

Update: *Acknowledged, not resolved. The Contracts team stated:*

> *In `_cloneCodeWithImmutableArgs`, we are building a bytes object. We decided to use a more "natural" solidity version that the compiler is free to compile to whatever it wants. The compiler may decide to use mcopy is the targeted EVM version supports it.*
>
> *`Bytes.slice` is a bit different because solidity does not provide a high level version of it. The few choices we have are:*
>
> - *writting a for loop in solidity*
> - *writting a for loop in assembly*
> - *using the identity precompile*
> - *using mcopy*
>
> *We believe that the first two options are are not optimal. Also, because using the identity precompile is more expensive than using mcopy, we decided to use mcopy. This indeed prevents using this code with a target older than Cancun. Having a second version of the function (that uses the precompile) is not really a possibility because:*
>
> - *a compiler would refuse to compile Bytes.sol with a version before Cancun, even if `slice` is not used and only `slicePrecompile` is called by the user.*
> - *we would have to duplicate all implementation that call that function directly or indirectly.*
>
> *We believe the impact of this issue is limited. Only the new code uses Bytes.sol, so this implementation is not breaking anything older. The pragma requires using of the recent version of the compiler (as referenced in our "documentation"), and the compiler throws very clear errors if a user tries to compile this code with an older EVM target. Ultimately, using an unsupported target should be identified by any serious testing done by the user.*

L-02 Possible Incorrect Updates to Voting Units Balance Checkpoints

The `_transferVotingUnits` function of the `Votes` contract is in charge of transferring, minting, and burning voting units. It can be used by contracts extending `Votes` to track changes in the distribution of these units. For example, `_transferVotingUnits` is called within `ERC20Votes` immediately after a transfer. However, in `VotesExtended`, `_transferVotingUnits` is overridden to update the `_balanceOfCheckpoints` mapping using the `_getVotingUnits` function to create a new checkpoint. This introduces a key difference in how `_transferVotingUnits` should be used in derived contracts compared to `Votes`. For instance, if an ERC-20 contract extends `VotesExtended`,

`_getVotingUnits` will produce different values depending on whether it is called before or after the ERC-20 balance update. Thus, in `VotesExtended`, `_transferVotingUnits` must be called after the balance changes to ensure accurate checkpoint updates, while in `Votes`, it can be used before or after without any issue.

To prevent misuse, consider comprehensively documenting the aforementioned distinction, emphasizing the order of execution of the internal functionalities.

Update: *Resolved in pull request #5306 at commit 334b617.*

L-03 Different Pragma Directives

In order to clearly identify the Solidity version with which the contracts will be compiled, pragma directives should be consistent across file imports. Throughout the codebase, multiple instances of varying pragma directives being used were identified:

- `CAIP10.sol` has the `pragma solidity ^0.8.24;` pragma directive and imports `Strings.sol`, which has a different pragma directive.
- `CAIP2.sol` has the `pragma solidity ^0.8.24;` pragma directive and imports `Strings.sol`, which has a different pragma directive.
- `Bytes.sol` has the `pragma solidity ^0.8.24;` pragma directive and imports `Math.sol`, which has a different pragma directive.

Consider using the same pragma version in all files.

Update: *Resolved. The Contracts team stated:*

> *`Bytes.sol` uses `mcopy` which is only supported since 0.8.24. This is why `Bytes.sol` (and `CAIP2.sol` and `CAIP10.sol` that depends on `Bytes.sol`) all require ^0.8.24 when other files are less restrictives.*
>
> *Note that `ReentrancyGuardTransient.sol`, `draft-ERC20TemporaryApproval.sol` and `TransientSlot.sol` have a similar requirement because of tstore/tload being introduced in 0.8.24*

L-04 Override Votes Do Not Count As Having Voted

The `GovernorCountingOverridable` governance module enables users (delegators) who have delegated their voting units to another account (delegate) to override that account's

decision. This is achieved by tracking voting units and delegates over time for each account. As a result, the delegator can override the delegate's decision by reallocating their voting weight, effectively reducing the delegate's voting power based on the proposal's snapshot.

However, when a user casts an override vote, their support is not tracked in the `casted` field of the `VoteReceipt` struct. Hence, when evaluating `casted` to determine whether the user has voted, the returned value will be `false`. This is misleading and can be misinterpreted by extending contract logic. Furthermore, it is worth noting that the `overridenWeight` field in the receipt is not tracked if a delegate has already voted, although this is not a problem as long as there is no getter and the mapping remains private.

Consider moving the logic of the `hasVoted` function into a new `hasVotedDelegated` function and redefining `hasVoted` to return the boolean `OR` of `hasVotedDelegated` and `hasVotedOverride`.

Update: *Resolved in pull request #5309 at commit 1c762d8. The `hasVoted` logic remains, while documentation was added. The Contracts team stated:*

> *The design of `GovernorCountingOverridable` purposely separates regular votes from overridden votes. The rationale is that both workflows are parallel.*
>
> *For example, Alice may be a delegate for Bob and Charles while Alice herself could delegate to Daniel. In this scenario, Alice can override Daniel's delegation (Alice own tokens) regardless of Daniel's vote, whereas Alice could vote with the delegated power of Bob and Charles.*
>
> *We are documenting this behavior more thoroughly. However, we consider merging both delegation votes and overridden votes in the `hasVote` function may be an issue for off-chain integrations that may assume that the user already voted after an override, when they can still cast their vote as in the case of Alice.*

L-05 Potentially Incorrect Hashing of User Operations

The `hash` function of the `ERC4337Utils` library is intended to compute the hash of a user operation using the ERC-4337 entrypoint's address and the chain ID. However, the variant of this function that only takes the user operation as input uses `address(this)` for the entrypoint's address, although the library is not designed to be used only by entrypoint implementations. As a result, hashes generated by this function will be invalid if generated by non-entrypoint contracts, leading to accounts rejecting signatures based on such hashes.

Consider having the entrypoint's address as a parameter in both variants of the `hash` function.

Update: *Resolved in pull request #5308 at commit 1816bb2. The `hash` function specifying `address(this)` as the entrypoint was removed.*

L-06 Nonce Key Not Included in `InvalidAccountNonce`

The `NoncesKeyed` contract extends `Nonces` to add support for keyed nonces, where each nonce is composed of a "key" and a "sequence". In the `_useCheckedNonce` function, `NoncesKeyed` reuses the inherited `InvalidAccountNonce` error. However, this error does not include details about the specific key causing the revert, thereby hindering code auditability.

Consider implementing a new `InvalidAccountNonce` error that includes information about the key that was responsible for the failure.

Update: *Resolved in pull request #5312 at commit d977260. The key is now prepended to the sequential nonce when reverting with the existing `InvalidAccountNonce` error.*

L-07 Inconsistent Format in Returned Nonces

In the `NoncesKeyed` contract, both the `nonces` and `_useNonce` functions return the next unused nonce for a given address and key. However, the `nonces` function prepends the key to the sequential nonce, while `_useNonce` does not. The `_useNonce` function is also commonly used in EIP-712 type hashes (e.g., `ERC20Permit`). Using the nonce without prepending the key would be less intuitive in this context and even raises replayability concerns.

To improve consistency, consider always returning the nonce with the prepended key.

Update: *Resolved in pull request #5312 at commit d977260.*

L-08 Misleading and Incomplete Documentation

Throughout the codebase, multiple instances of misleading documentation were identified:

- The documentation for the `_useCheckedNonce` function states that it accepts a nonce as a single `uint256` parameter, with the first 8 bytes representing the key and the remaining 24 bytes representing the nonce. However, the key is actually 24 bytes long, while the nonce is 8 bytes.
- The documentation for the `cloneDeterministicWithImmutableArgs` function states that multiple clones under the same implementation address and salt will revert. This must not be true because the immutable arguments become part of the bytecode and thereby influence the deployed address. Thus, it is possible to reuse the same implementation address and salt provided that the immutable arguments are changed.
- Within the `PackedUserOperation` struct, the `maxPriorityFeePerGas` is referred to as `maxPriorityFee`.
- The `paymasterAndData` field of the `PackedUserOperation` struct does not indicate the size of the `paymasterVerificationGasLimit` and `paymasterPostOpGasLimit` data, while the getter functions suggest that they are encoded with 16 bytes each.
- The term "delegatee" should be replaced with "delegator" when referring to users who delegate their voting power. This is because "delegatee" actually means the recipient (or delegate) rather than the initiator of delegation. This adjustment should be made in the docstring for `GovernorCountingOverridable` and in the `_delegateCheckpoints` mapping.

Consider correcting the aforementioned comments to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #5310 at commit ff30cd4 and pull request #5324 at commit 737d0e1.*

L-09 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `GovernorCountingOverridable.sol`:
  - The `OVERRIDE_BALLOT_TYPEHASH` state variable
  - The `VoteReduced` event
  - The `OverrideVoteCast` event
  - The `GovernorAlreadyOverridenVote` error

- In `draft-IERC7579.sol`:
  - The `IERC7579Module` interface
  - The `IERC7579Validator` interface
  - The `IERC7579Hook` interface
  - The `IERC7579Execution` interface
  - The `IERC7579AccountConfig` interface
  - The `IERC7579ModuleConfig` interface
  - The `ModuleInstalled` event
  - The `ModuleUninstalled` event

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #5311 at commit fa9a059. The Contracts team stated:*

> *We are not adding documentation for `OVERRIDE_BALLOT_TYPEHASH`, which is consistent with other typehashes across the library. Similarly, we are not documenting the `GovernorAlreadyOverriddenVote` error given its clarity.*

L-10 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- In `ERC1363.sol`:
  - In the `transferAndCall` function, the return value is not documented.
  - In the `transferFromAndCall` function, the return value is not documented.
  - In the `approveAndCall` function, the return value is not documented.
- In `GovernorCountingOverridable.sol`:
  - In the `proposalVotes` function, the `proposalId` parameter and the return value are not documented.
  - In the `castOverrideVote` function, the `proposalId`, `support`, and `reason` parameters and the return value are not documented.
  - In the `castOverrideVoteBySig` function, the `proposalId`, `support`, `voter`, `reason`, and `signature` parameters and the return value are not documented.

- In `draft-ERC7579Utils.sol`:
  - In the `ERC7579TryExecuteFail` event, the `batchExecutionIndex` and `result` parameters are not documented.
- In `draft-IERC7579.sol`:
  - In the `validateUserOp` function, the return value is not documented.
  - In the `executeFromExecutor` function, the return value is not documented.
- In `draft-IERC4337.sol`:
  - In the `validateUserOpSignature` function, the return value is not documented.
  - In the `handleOps` function, the `beneficiary` parameter is not documented.
  - In the `handleAggregatedOps` function, the `beneficiary` parameter is not documented.
  - In the `validateUserOp` function, the `userOp`, `userOpHash`, and `missingAccountFunds` parameters and the return value are not documented.
  - In the `validatePaymasterUserOp` function, the `userOp`, `userOpHash`, and `maxCost` parameters and the return values are not documented.
  - In the `postOp` function, the `mode`, `context`, `actualGasCost`, and `actualUserOpFeePerGas` parameters are not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #5315 at commit 2eb856e and pull request #5324 at commit 10c7594.*

Notes & Additional Information

N-01 Incomplete Account Utility Libraries

The `ERC7579Utils` and `ERC4337Utils` libraries appear to be incomplete. The following opportunities for completion were identified:

- The ERC-7579 standard expects call types to execute a call, batch call, static call, or delegatecall. However, the static call type (`0xFE`) is not listed among the other call type constants.
- The `ERC4337Utils` library provides getter functions to extract packed information from the `PackedUserOperation` struct. However, there are no functions to extract the `factory` address and `factoryData` from the `initCode` field, or `paymasterData` from the `paymasterAndData` field.

To improve developer utility, consider adding the aforementioned missing functionality to the utility libraries.

Update: *Resolved in pull request #5313 at commit 2eb1be1. The Contracts team stated:*

> *The use case for the 0xFE call type in ERC-7579 is unclear. This may come in a further version but we did not want to commit to this in 5.2 version.*
>
> *The additional ERC4337Utils features were implemented in pull request #5313. This also included improving the existing paymaster getters in case the `paymasterAndData` field is empty.*

N-02 Duplicated Logic

In the `Votes` and `VotesExtended` contracts, the logic to validate a given timepoint against the current timepoint is repeated across four functions: `getPastVotes`, `getPastTotalSupply`, `getPastDelegate`, and `getPastBalanceOf`.

Consider consolidating this duplicated logic into a reusable function to improve consistency and readability.

Update: *Resolved in pull request #5314 at commit eb51fbc.*

N-03 Constant Visibility Not Explicitly Declared

Within `draft-ERC7579Utils.sol`, multiple instances of constants lacking an explicitly declared visibility were identified:

- The `CALLTYPE_SINGLE` state variable
- The `CALLTYPE_BATCH` state variable
- The `CALLTYPE_DELEGATECALL` state variable
- The `EXECTYPE_DEFAULT` state variable
- The `EXECTYPE_TRY` state variable

For improved code clarity, consider always explicitly declaring the visibility of constants, even when the default visibility matches the intended visibility.

Update: *Resolved in pull request #5308 at commit fa30a20.*

N-04 Unused Imports

Throughout the codebase, multiple instances of unused imports were identified:

- In `draft-ERC4337Utils.sol`, the `IEntryPoint` import is unused.
- In the `CAIP2` and `CAIP10` libraries, the `SafeCast` library is unnecesarily imported and used for the `uint256` type.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #5308 at commit 4c5935a.*

N-05 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In line 79 of `ERC4337Utils.sol`, "Sames" should be "Same".
- In line 7 of `NoncesKeyed.sol`, "support" should be "supports".
- In line 22 of `NoncesKeyed.sol`, "this functions" should be "this function".
- In line 60 and line 166 of `Clones.sol`, "multiple time" should be "multiple times".
- In line 14 of `draft-IERC4337.sol`, "bunder" should be "bundler".
- In line 9 of `VotesExtended.sol`, either "adds" or "exposes" should be removed.
- In line 12 of `GovernorCountingOverridable.sol`, one "token" should be removed.

Consider fixing the typographical errors to improve the readability of the codebase.

Update: *Resolved in pull request #5308 at commit 1fa2532.*

N-06 Redundant Code

Throughout the codebase, multiple instances of redundant code were identified:

- The `unchecked` keyword in the `indexOf` function is redundant. This is because since Solidity version 0.8.22, the compiler itself optimizes the loop increment.
- The `Math.ternary` operation in the `gasPrice` function is redundant. This is because if `maxFee` and `maxPriorityFee` are equal, `maxFee` will be returned by the `Math.min` operation anyway.

Consider removing redundant code to enhance the clarity and efficiency of the codebase.

Update: *Resolved in pull request #5308 at commit 15a00f5.*

N-07 Code Clarity

Throughout the codebase, multiple opportunities for improving code quality were identified:

- Using decimal notation instead of hexadecimal would improve the readability and ease of validation for byte amounts and offsets. For instance:
  - In the `ERC4337Utils` library, lines 29, 113, and 123.
  - In the `Clones` library, lines 230, 232, and 251. Furthermore, in line 232, the `0x20` constant could be replaced with `32` for consistency.
- The `SIG_VALIDATION_SUCCESS` constant can replace the magic number `0` for checking validation data.
- There is a `_delegateCheckpoints` mapping in the `VotesExtended` contract and in the `Votes` contracts that is extended. However, this mapping serves two different purposes: in `Votes`, it tracks the amount of delegated votes per address over time, whereas in `VotesExtended`, it tracks which address delegated to whom over time. Consider renaming these mappings to `_delegateVotesCheckpoints` and `_chosenDelegateCheckpoints` or similar, respectively, to clarify their distinct roles.
- The `_balanceOfCheckpoints` mapping in `VotesExtended` is misleading as it does not track token balances per account. Instead, it tracks the voting units defined by `_getVotingUnits`. Since balance and voting units do not necessarily have a 1:1

relationship, consider renaming the mapping to `_votingUnitsCheckpoints` or a similar term.
- The rationale for deriving `NoncesKeyed` from `Nonces` and using the inherited functionality to handle key `0` should be clarified.

Consider incorporating the above-listed changes into the codebase to enhance code clarity and readability.

Update: *Resolved in pull request #5317 at commit 7a3707f.*

Client Reported

CR-01 Case-Sensitivity in CAIP-10 Identifiers

During the audit, the contracts team raised concerns about the ambiguity in CAIP-10 identifiers when it comes to the case sensitivity of Ethereum addresses, as introduced in EIP-55. The Canonicalization section of the standard states that, to date, this canonicalization is not required and remains at the developers' discretion. However, case sensitivity in CAIP-10 strings can create issues in contexts like hashing, where they may serve as mapping keys. In such cases, lowercase and checksummed addresses are semantically equivalent but produce different slot values.

Consider documenting a warning about the implications of case sensitivity in the `CAIP2` and `CAIP10` libraries. In addition, consider adding a `toLowercase` function to the `Strings` library to provide developers with a tool to manage this ambiguity.

Update: *Resolved in pull request #5319 at commit 133adf8. The Contracts team stated:*

> *We are documenting the issues that may arise from the lack of canonicalization of the identifiers. For a consistent representation, we recommend using the `toChecksumHexString` function.*

Conclusion

The v5.2 release of OpenZeppelin Contracts introduces several utility libraries, primarily focused on supporting account abstraction and cross-chain messaging. In addition, a new governance module has been developed, empowering users to override the voting power of their delegates. We commend the Solidity Contracts team for addressing user needs and enhancing existing features while introducing valuable new utilities.

During the audit, particular care was taken to document edge cases, ensuring that integrators are informed of potential risks when interacting with these contracts. Such efforts aim to create a more resilient codebase, recognizing the library's critical role as a foundational component within the blockchain ecosystem. The Contracts team has demonstrated a strong commitment to maximizing the library's security and we are glad to have collaborated with the Contracts team on this milestone.